Modelling and Simulation of Standardised Control Functions from Building Automation

Georg Ferdinand Schneider[1] Georg Ambrosius Peßler[1] Simone Steiger[1]

[1]Group Technical Building Systems, Fraunhofer Institute for Building Physics IBP, Nürnberg, Germany,
georg.schneider@ibp.fraunhofer.de, georg.pessler@ibp.fraunhofer.de,
simone.steiger@ibp.fraunhofer.de

Abstract

Despite the accepted fact that control logic deployed in future and existing buildings through building automation systems constitutes a key factor for increasing their energy efficiency, the support for modelling and simulation of these in current state-of-the-art simulation tools and libraries is rather limited. In particular a gap exists for modelling and simulation of standardised control functions. In this work we present an approach for modelling standardised control logic using Modelica. We evaluated the interoperability of the modelling approach by simulating a test case of an automation solution controlling the sunshade of a room and by reimplementing a state-based control for an air handling unit reusing models from two Annex60 compliant libraries.

Keywords: Building Automation, Control Function, VDI 3813, VDI 3814, ISO 16484

1 Introduction

The three pillars which influence the energy demand in buildings are a sophisticated façade, energy efficient technical equipment and the actual operation by means of control through a Building Automation System (BAS). The use of BAS is stipulated by relevant standards, e.g. EN 15232:2013. However, benefits in terms of reduced energy demand from the façade and/or technical equipment can easily be spoilt by operating a building using a poorly designed, misconfigured or malfunctioning BAS.

The operation of a building is a complex control task involving multiple sensors, actuators and control algorithms spanning different scales in terms of time and space; the sum of input and outputs can easily reach ten thousand. Hence, the design, (continuous-) commissioning and operation of such a complex system is a challenging, time- and cost-intensive task.

A possible solution for managing this complexity during BAS design and operation is the use of a Model-Based Design (MBD) methods, where all components of a building are modelled and simulated to design and test the control logic of a BAS prior to its deployment in a building. Also, comparison of the simulation model and the real-world implementation provides a helpful insight in detecting anomalies during operation (Venkatasubramanian et al., 2003). The model and adjacent simulation infrastructure can further be used for Model-in-the-loop and Software-in-the-Loop (SIL) evaluation, e.g. for automotive applications (Chrisofakis et al., 2011) and later in Hardware-in-the-loop simulation, e.g. for circulating pump control (Schneider et al., 2015).

Models to describe the behaviour of control logic and algorithms are part of the Modelica Standard Library (MSL) since its very beginnings. A research effort from the International Energy Agency's Energy Buildings and Communities Programme (IEA EBC) develops as one outcome a core library for Building Performance Simulation (BPS) using Modelica. A set of four libraries, all suitable for MBD within the buildings domain, `Buildings` (Wetter et al., 2014), `AixLib` (Constantin et al., 2014) `BuildingSystems` (Nytsch-Geusen et al., 2013) and `IDEAS` (Baetens et al., 2012) now share one common core library `Annex60` (Wetter et al., 2015). A special library `NCLib` for the simulation of automation systems is presented by Liu (2013), however its focus is on modelling automation system devices rather than the control logic involved. A library specifically designed to model and simulate control functions from industrial automation in Modelica without specific control functions for standardised room or building automation is presented by Bonvini and Leva (2012).

Several national and international standards exist to support the design process of BAS, e.g. VDI 3813-2:2011; VDI 3814-6:2009; ISO 16484:2011 by providing a set of commonly utilised control functions which can be reused to compose an automation solution for room and building automation. The number of defined control functionalities and the detail of the descriptions varies between standards. Having these pieces of control logic readily available in a simulation environment to compose by drag-and-drop a control strategy for a room or a piece of equipment in a building can result in significant benefits in terms of time required for the design and quality of the resulting control logic. Automation engineers may design and test their control solution prior to the deployment in a real building by coupling its simulation to models of rooms, buildings and equipment. Also in later stages of the life-cycle the outcome of the simulation model can be compared with actual monitoring data for fault detection purposes.

However, to the best of our knowledge, no library exists for the modelling and simulation of standardised control functions (see comparison in Table 1). The contribution aspect of this work resides in presenting Modelica-based modelling approach for standardised control functions from BAS.

We present a model library `BuildingControlLib` which provides a basis to implement standardised control functions in a streamlined manner. For the implementation of block oriented control functions from standards we recommend representing the structure by class diagrams and the actual control behaviour for state-based control logic using activity diagrams, as defined by the Unified Modeling Language (UML) (Object Management Group, 2015). The design of the models and library is such that the compatibility to the MSL as well as libraries from Annex60 effort is ensured. The graphical visualisation included in the models is designed such that representations composed to the standards and control solutions may be compiled in a ease-to-use manner from interested BAS practitioners.

As a beginning we include models of control functions compliant to the standards VDI 3813-2:2011 and VDI 3814-6:2009. Due to partly ambiguous textual descriptions for control behaviour in standards the models compliant to VDI 3813-2:2011 are designed such that standardised interfaces and actual control functionality are separated. This offers the benefit that the actual functionality can be easily exchanged with own code or implementations, possibly using the Functional Mockup Interface (FMI) standard (Blochwitz et al., 2012). To support users when composing own control solutions by drag and dropping control functions from the library we include the notion of connector semantics (Dibowski et al., 2010) to help users composing only semantically correct automation solutions; for example it is not possible to connect an indoor air temperature output and an outdoor air temperature input. To represent state-based control we include models to simulate state-based control descriptions in BAS as defined in VDI 3814-6:2009

In the remainder of this work we describe the underlying design principles when modelling standardised control functions in section 2. We then demonstrate the usability of the approach by simulating two test cases where the automation models are coupled to physical room and equipment models from Annex60 compliant libraries in section 3.

2 Implementation

The models are included in a library termed `BuildingControlLib`. Its overall structure is depicted in Figure 1 from a screenshot in Dymola 2015 FD01 (Dymola, 2015) which we use for implementation. The design of the overall structure follows the best practices and conventions documented in the MSL, e.g. naming convention of models and classes, package structure with a user's guide, ready-to-simulate examples, components, interfaces and types.

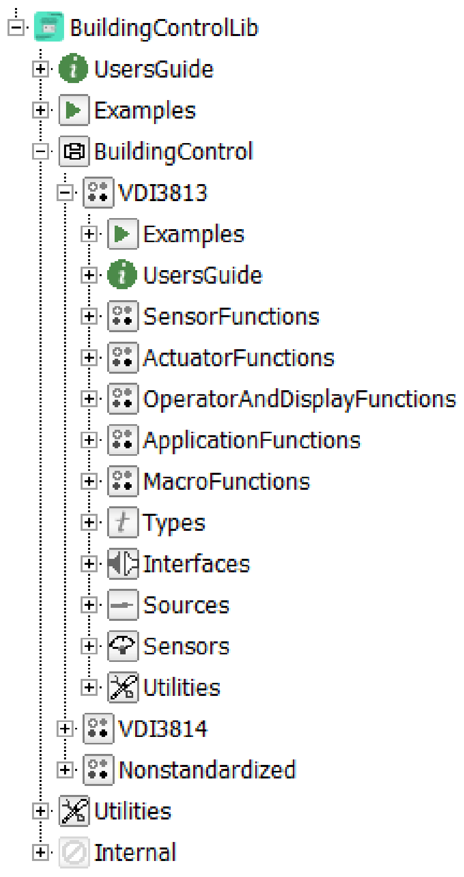

Figure 1. Overall structure of model library.

At the top level we include the mandatory packages for documentation and examples and three packages `VDI3813`, `VDI3814` and `Nonstandardized`. The number of packages mentioned is not meant to be exhaustive. Instead the implementation undertaken so far may serve as a blue print for implementing control functions from further standards, e.g. ISO 16484:2011.

The first package contains models for the simulation of room automation functions from VDI 3813-2:2011; the second contains models for the simulation of state graphs as defined in VDI 3814-6:2009; and the third package gathers models for common non-standardised control functions. In some cases, e.g. a schedule, we reuse functionality already implemented for non-standardised applications.

Auxiliary models used in example models are kept in the `Utilities` package.

2.1 Room Automation According to VDI 3813

Beside the general best practices for Modelica libraries, e.g. package structure, the following requirements have been defined to enable seamless use of the library:

1. Modular design such that control functionality is encapsulated and may be exchanged as needed;

Table 1. Overview of the reviewed open-source libraries. X - criteria fulfilled and - not fulfilled. (1) - `Annex60` (Wetter et al., 2015), (2) - `Buildings` (Wetter et al., 2014), (3) - `AixLib` (Constantin et al., 2014), (4) - `BuildingSystems` (Nytsch-Geusen et al., 2013), (5) - `IDEAS` (Baetens et al., 2012), (6) `NCLib` (Liu, 2013), (7) - `IndustrialAutomationSystems` (Bonvini and Leva, 2012) and (8) - this work.

Criteria	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Models for								
... control	X	X	X	X	X	-	X	X
... room automation	(X)	(X)	(X)	(X)	(X)	-	-	X
... building automation (BA)	(X)	(X)	(X)	(X)	(X)	-	-	X
... standardised BA	-	-	-	-	-	-	-	X
Semantic connectors	-	-	-	-	-	-	-	X
Based on Annex60	obsolete	X	X	X	X	-	-	-
Active development	X	X	X	X	X	-	X	X

2. Automated compatibility checking of control functions as proposed by Dibowski et al. (2010);

3. Graphical representation as defined within the standard.

The first requirement results from prevalent heterogeneity in actual implementations of control behaviour in standards. Existing standards do provide textual descriptions on how the actual behaviour should be, however this descriptions leave room for interpretation. Thus functionalities might comply to standards but have minor differences. To ensure the seamless exchange of functionality, possibly from non-Modelica implementations, we separate the definition of interfaces (function) from its functionality as described in detail in the next sections.

The second requirement is motivated by an approach reported by Dibowski et al. (2010). The methodology described allows to automatically derive interoperable automation solutions for room automation. The approach relies on the formal specification of input and output variables of control functions, by annotating exchange variables with information on e.g. its unit, quantity, etc. The approach is exemplified for control functions for room automation (Dibowski, 2013). As Modelica provides mechanisms for consistency checking on exchange variables the requirement should be fulfilled by implementation to support library users when implementing own solutions.

Finally to allow easy composition of control solutions also by interested practitioners the visual appearance of the modelled control functions should align with the definitions in the standards. Thus allowing the easy reuse and composition of automation solutions.

Structure of `VDI3813` Package

The top-level structure of `VDI3813` package is illustrated in Figure 1. It follows the taxonomy established within the standard which classifies available control functions into:

- `SensorFunctions` - which convert physical signals into automation signals;

- `ActuatorFunctions` - which receive a setpoint to generate a physical control command for a motor;

- `OperatorAndDisplayFunctions` - which exchange status information to occupants and give them the ability to send manual commands;

- `ApplicationFunctions` - which provide the actual automation functionality by processing sensor or operator functions and transmit new setpoints and commands to actuators;

- `Macrofunctions` - which provide an interface to compose reusable macro-functions from low-level control functions.

The standard also defines supervisory control functions such as data storage and external messaging which we found to be out of scope for dynamic simulation of control behaviour and coupling to models of physical processes. The category *Common I/O functions* with two control functions for interfacing the automation system to external applications is not explicitly modelled.

To ensure computationally efficiency we implement the control functions as Modelica `block` as suggested by Liu (2013). Whenever possible we reuse models from MSL.

Encapsulating Functionality

The actual functionality in the standard is defined using textual descriptions. This leaves a wide range for interpreting and implementing this descriptions; hence, implementations of control functions vary between different manufacturers of devices for room automation.

To represent and model this heterogeneity as defined in the first requirement (see section 2.1) the following design principle is applied within this library. We introduce a block `Function` as a base class for defining the interface to other functions. Each `Function` has a associated block `Functionality` (see Figure 2) which serves as a template to implement the respective intended functionality. This allows for easy maintainability and quick exchange of functionality within the library, e.g.

a manufacturer would like to place his own functionality within the model, potentially using the FMI standard (Blochwitz et al., 2012). By leaving the respective block `Function` unaltered its interoperability with other function blocks is ensured and changes are only applied within `Functionality`.

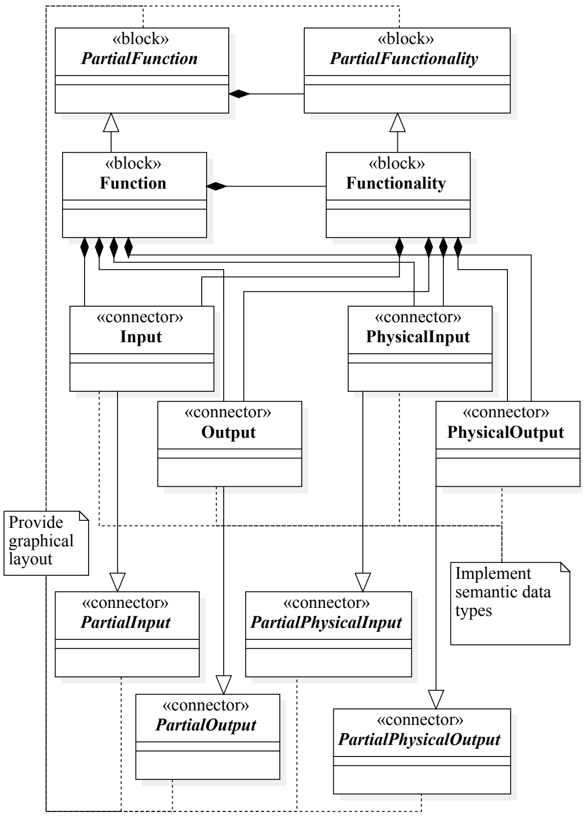

Figure 2. Class diagram in UML describing the modelling principle of encapsulating functionality.

To ensure uniform and complaint graphical layout of the control functions it is once defined in the partial block `PartialFunction` using Modelica annotations.

To implement a control function a block is created which inherits from `PartialFunction`. Parameters and connectors must be added. In the corresponding functionality block the actual control logic is implemented in what ever way is preferred, e.g. reusing models from other libraries.

For all implemented control functions we provide a default functionality which may be adapted to the users needs or exchanged if required. However, regarding the mentioned space for interpretation arising from textual descriptions in the standard these are not meant to be normative. To ensure understandability we provide UML activity diagrams for describing the respective functionality as implemented and include it into the documentation of each model. As an example we present the UML activity diagram of `AutomaticThermalControl` in Figure 3.

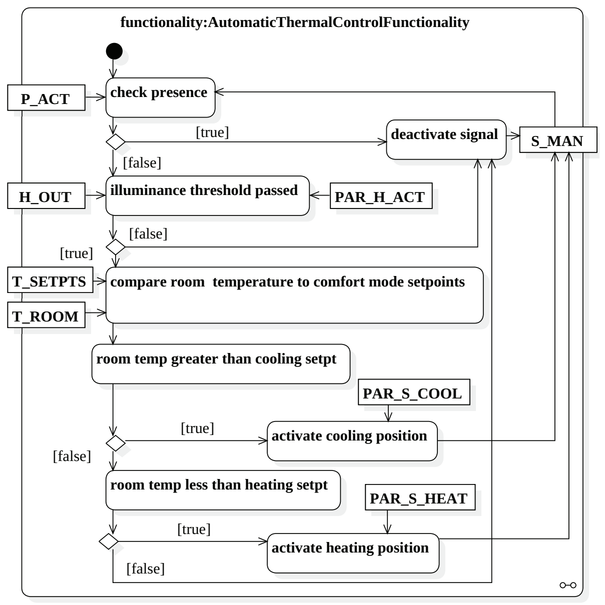
functionality:AutomaticThermalControlFunctionality
P_ACT
check presence
[true]
deactivate signal
S_MAN
[false]
H_OUT
illuminance threshold passed
PAR_H_ACT
[false]
[true]
T_SETPTS
compare room temperature to comfort mode setpoints
T_ROOM
room temp greater than cooling setpt
PAR_S_COOL
[true]
activate cooling position
[false]
room temp less than heating setpt
PAR_S_HEAT
[true]
activate heating position
[false]

Figure 3. UML activity diagram of a functionality `AutomaticThermalControl` as described in VDI 3813-2:2011.

Semantic Connectors

To solve the task of automating the design process of room automation systems recently the semantically unambiguous specification of function profiles for room automation is introduced (Dibowski et al., 2010; Dibowski, 2013). This approach allows for the automated generation of room automation profiles, i.e. a set of control functions from a standard, automatically from initially defined requirements. The approach requires to formally specify automation devices including their functional profiles, i.e. the control functions implemented on a devices. Moreover to automatically bind variables of different control functions detailed semantics of the input and output variables are specified. The approach has been successfully demonstrate by modelling functions and devices complying to the VDI 3813-2:2011 standard.

To support library users in the design of a room automation solution using the library we integrate the notion of semantics in the design of connector classes. We define for every variable type in the standard a separate connector class. We specify the unit, quantity, basic type (`Real`, `Boolean`, `Integer`, ...) and direction of information flow (`input`/`output`) in the connector definition and a corresponding type definition using the known Modelica language elements for this. As emphasised by Dibowski et al. (2010); Dibowski (2013) these specifications are not enough to differentiate among input and output variables. As Modelica does not provide additional ways to specify variable semantics we introduce a naming convention for exchange variables specified in the connector classes. The naming convention has two parts: (1) Each variable starts with one of the strings *value*, *status*, *command* and *setpoint*, a classification recommended by Dibowski (2013);

(2) in the second part we added additional strings specifying additional semantics e.g. to differentiate between an indoor and an outdoor air temperature.

For example for the exchange variable describing the outdoor illuminance abbreviated in the standard as *H_OUT* we specify a connector ValueIlluminanceOutdoor:

Listing 1. Source code of connector `ValueIlluminanceOutdoorInput`.

```
connector ValueIlluminanceOutdoorInput
   extends Partial.PartialInput;
  input
     BuildingControlLib.[ ... ].
         ValueIlluminanceOutdoor
   valueIlluminanceOutdoor; // Specified
       variable name
end ValueIlluminanceOutdoorInput;
```

The semantic correctness when composing automation solutions from drag and dropping control function blocks in e.g. a macro is ensured by the ability of a Modelica simulation environment to check connector compatibility in terms of unit, quantity, basic type, input/output and the name of the variable. Hence, a user can only connect different control functions if input and output connectors match. To ensure compatibility with models which implement connectors using the MSL interfaces, converter models are provided in the `Sources` and `Sensors` packages.

2.2 State Graph According to VDI 3814

Control logic for the control of Heating Ventilation and Air-Conditioning (HVAC) often follows a state-centric behaviour. Multiple descriptions exist for modelling this behaviour originally described by Harel (1987). The standard VDI 3814-6:2009 defines the concept of a *State Graph* to provide a graphical representation of state-centric control behaviour in BAS.

In the package `VDI3814` we provide models to compose by drag-and-drop a State Graph. The graphical layout of the models fits the definitions in the standard (see Figure 4). For implementing we use models from `Modelica.StateGraph` package from MSL.

An example of the modelling capabilities of the package is displayed in Figure 4 where the control of a generic air handling unit is modelled with control states off, cooling, heating or frost protection. A specific characteristic of VDI 3814 State Graphs is that the explicit concept of a transition does not exist. A new state is active when a Boolean expression is evaluated as true. However transition conditions are included into a state. This definition is modelled by including an array of transitions and a state into one model. This allows to display the designed state graphs as defined in the standard while reusing the models from `Modelica.StateGraph` package. Also in the standard it is possible to put *return* objects symbolised by a circle around a number of the targeted state. This can also be modelled and when removing the graphical anno-

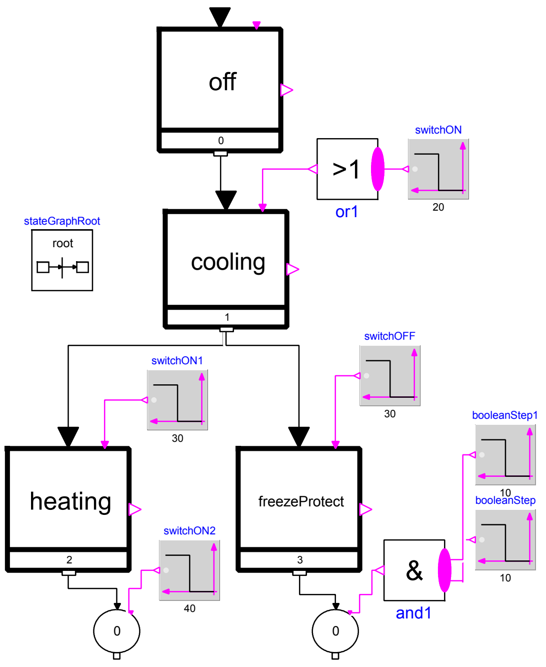

Figure 4. Ready to simulate model of a state graph from package `VDI3814`.

tations of the connect statements it is possible to derive a graphical layout which is very similar to the one used in the standard.

Additionally we reused blocks from MSL to model logical conjunction and logical disjunction as required in the standard.

3 Results from Test Cases

To evaluate the performance of the implemented control library and to examine its interoperability to existing Modelica libraries in the domain, we present results from two test cases implemented for demonstration purposes. In the first test case we implement an automation solution of a sunshade in a room using control function models from the package `VDI3813` where we reuse a room model included in the `AixLib`-library (Constantin et al., 2014). In the second example we re-implement the state-based control of an Air Handling Unit from `Buildings`-library (Wetter et al., 2014) using the models provided in the `VDI3814` package.

3.1 Room automation according to VDI 3813

Most room automation control functions focus on maintaining acceptable indoor comfort conditions for occupants while minimising the energy demand required to provide these comfort conditions to the occupant.

In Figure 5 the scheme of an automation solution for a room with a sunshade is illustrated. It is composed of control functions from VDI 3813-2:2011 which are modelled within the library presented in this work. As outlined above, the standard distinguishes between: sensor functions; actuator functions; operator and display functions and application functions. Disconnected inputs in the simulation of this test case have been fixed to reasonable constants for keeping results easy to understand.

The overall functionality intended to be realised here automatically limits or increases the amount of solar heat gains of a room either by deploying or elevating a sunshade, respectively. It is implemented as a macro function (VDI 3813-2:2011) using the application functions `OccupancyEvaluation`, `PriorityControl`, `AutomaticThermalControl` and `SetpointCalculation`. The connection to the physical inputs required by these functions is realised using the control functions `PresenceDetection` (Check with a sensor if room is occupied), `WindowMonitoring` (Sensor function to check if window is open), `BrightnessMeasurementOutdoor` (Determine outdoor brightness), `AirTemperatureMeasurementRoom` (Determine room air temperature) and `AirTemperatureMeasurementOutdoor` (Determine outdoor air temperature). A user may adjust the current temperature setpoint via the `AdjustTemperatureSetpoint` function. The outcome of the automation solution affects the actual (real or virtual) sunshade by using the function `sunshadeActuator`.

`AutomaticThermalControl` is only active if the room is unoccupied and the outdoor illuminance levels are higher than a threshold. Then, dependent on the comparison of current setpoint of the room and the measured room temperature, the sunshade is either deployed or elevated.

Between the control output of `AutomaticThermalControl` and its actual deployment via the `ActuateSunshade` control function finally, i.e. `PriorityControl` checks if no higher prioritised signal is found or the window is open. If the signal of `AutomaticThermalControl` is of current highest priority, it is then forwarded to the `SunshadeActuator` control function and deployed on the actual sunshade.

We implemented the described room automation solution using models from the previously described library. We couple it to a model which captures the physical behaviour of a room which we adapted from the model `ASHRAE140.Case900FF` from Constantin et al. (2014). The boundary conditions (weather, internal gains, etc.) remained unchanged. We adapted the ventilation schedule to 30 min once every day and introduced a constant heat flow rate of 500 W to heat the room in order to limit the outcome of these disturbances to the sunshade control on the automation solution. Also we modified the model of the south facing wall to contain a window with a sunshade which may be deployed from outside via a Boolean connector. We calculate the value of the outdoor illuminance assuming a constant factor of 2.49 between the value of irradation on a horizontal surface provided by the model *radOnTiltedSurf_Perez[5]*.

We simulated the coupled room and automation model for the first day in the weather file. We chose boundary conditions such that no presence is detected and no automatic or manual set point changes are applied. Hence the automatic control is active only when illuminance levels are sufficient. Results of the simulation are presented in Figure 6. Presented therein are the input signals affecting the `AutomaticThermalControl` control function, i.e. in the upper third the outdoor illuminance H which is compared within the control function with a threshold P_H. If the illuminance is too low, no control action happens. In the mid part the Boolean expressions indicating if `AutomaticThermalControl` is operating in heating y_{hea} or cooling mode y_{coo} and a signal telling the sunshade to be deployed or not are plotted (u_{sun}). In the lowest subplot in Figure 6 the outdoor T_{oa} and indoor air temperature T_{ra} are given and their respective heating ($T_{hea,s}$) and cooling ($T_{coo,s}$) set point. Also the air exchange rate AER is given, representing the ventilation scheme applied.

Given the described boundary conditions the control functionality allows to keep the room temperature within the bounds set by the heating and cooling set points, $T_{hea,s}$ and $T_{coo,s}$ respectively. Before ventilating (time < 43200 s) the sunshade is deployed several times when the control switches to cooling mode, triggered by the actual room temperature reaching the upper temperature set point at 24 degrees Celsius. The sharp decline during ventilation results in a period where the heating mode is active and the sunshade is elevated.

When the temperature recovers after ventilation with outdoor air the cooling mode is active until the automatic control enters the inactive mode when illuminance levels fall below the threshold specified (P_H).

3.2 Control of an Air Handling Unit via VDI 3814 compliant State Graph

To evaluate the models implemented in the package `VDI3814` we re-implement the model `ModeSelector` from the `Buildings`-library (Wetter et al., 2014). The model is used in the example `VAVReheat.ClosedLoop` to control a model of an Air Handling Unit which supplies conditioned air to a thermal zone. It encompasses six states representing the behaviour of the system (initial, unoccupied off, morning pre-cooling, morning warm-up, unoccupied night set back and occupied).

We simulate `VAVReheat.ClosedLoop` with the provided model and with the same control logic implemented using models from `VDI3814`-package. We use a Dymola 2015 FD01, 64bit with the following simulation settings:

- start time: 0 s, end time: 172800 s;
- solver: Esdirk23a - order 3 stiff;
- interval length: 600 s; Tolerance: 1e-6.

We compare the results of the two simulations by calculating the R squared value and the Mean Absolute Percentage Error (MAPE) between the respective results. The

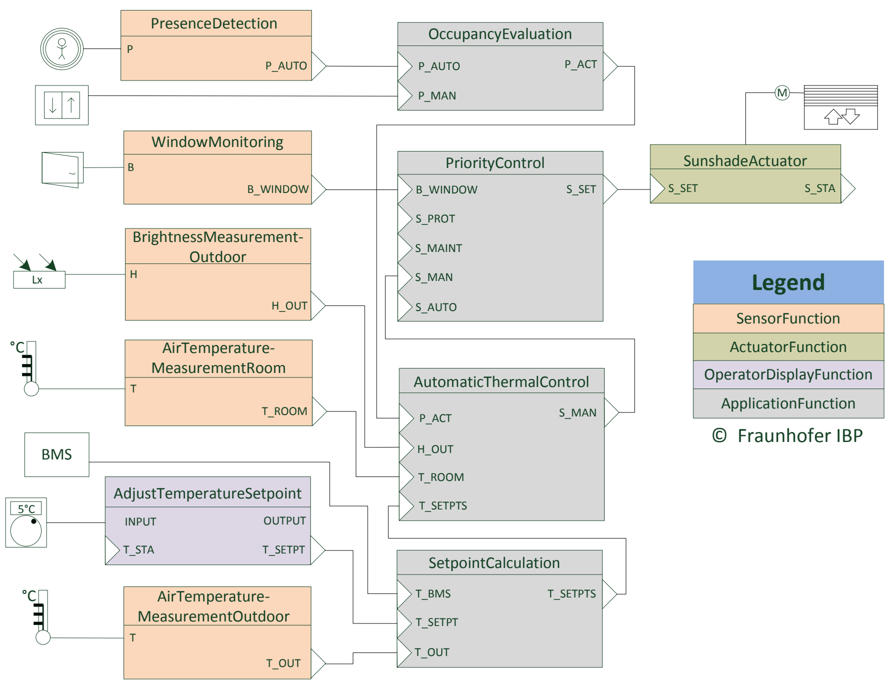

Figure 5. Scheme of the control functions utilised in the simulated room automation test case.

results are summarised in Table 2. The results show a high agreement of the two simulations reflecting the similar behaviour. Some discrepancies can be observed on the compared simulated data which can be explained from errors resulting from interpolating these values. An error of 0.192% for the control mode exists as a negligible deviation between the control mode signals of both simulation was identified; this originates from the waiting times which at some point in the state graph network need to be introduced. For the VDI3814 case with integrating transitions and states in one model, a fixed delay is introduced to avoid the termination of the simulation when initialising.

Table 2. Results from comparing the simulation of `VAVReheat.ClosedLoop` with the original model from `Buildings` library (Wetter et al., 2014) and this work. MAPE - Mean Absolute Percentage Error.

Variable	**R^2**	**MAPE** in %
`TOut`	.999	6.816e-6
`controlMode`	.970	0.192
`occupied`	1.00	0.000
`TRooMin`	0.999	6.475e-4
`TRooAve`	0.999	6.157e-4
`TRooSetCoo`	0.997	20.24e-4
`TRooSetHea`	0.997	19.48e-4

4 Discussion

From the results presented in this work the Modelica modelling language is found to be suitable for the proposed modelling and simulation of standardised control functions.

However, some limitations have been identified when implementing the notion of semantic connectors. We found the ability of the Modelica language to support checking of the consistency of connector variables according to units, quantity and the name of the variable to be a helpful feature. However, when attempting to include detailed semantics of connector variables the only possibility is to introduce a naming convention for the exchange variables which is cumbersome to maintain and prone to errors. It may be of interest to extend the Modelica language in future releases in this direction to evaluate the types of a connector variable (not only the basic type, e.g. `Real`, `Boolean`, `Integer`, etc.) allowing to define a taxonomy of types instead of a naming convention. An approach known from the simulation of cyber-physical systems embedded in the Ptolemy II framework offers a possibility to include consistency checking (Leung et al., 2009) of units based on ontology. The concept of `expandable connector` is not found to be suitable in this context as a variable name might occur several times in one automation solution but must not be connected to each of its occurrences.

Standards are evaluated and revised on a regular basis, thus a regular revision and maintenance of the library is required. We are confident that the presented underlying design principles of the modelling approach remain relevant and applicable to future and upcoming versions of standards.

Most standards provide textual descriptions of the functionality of a control function which often is ambiguous

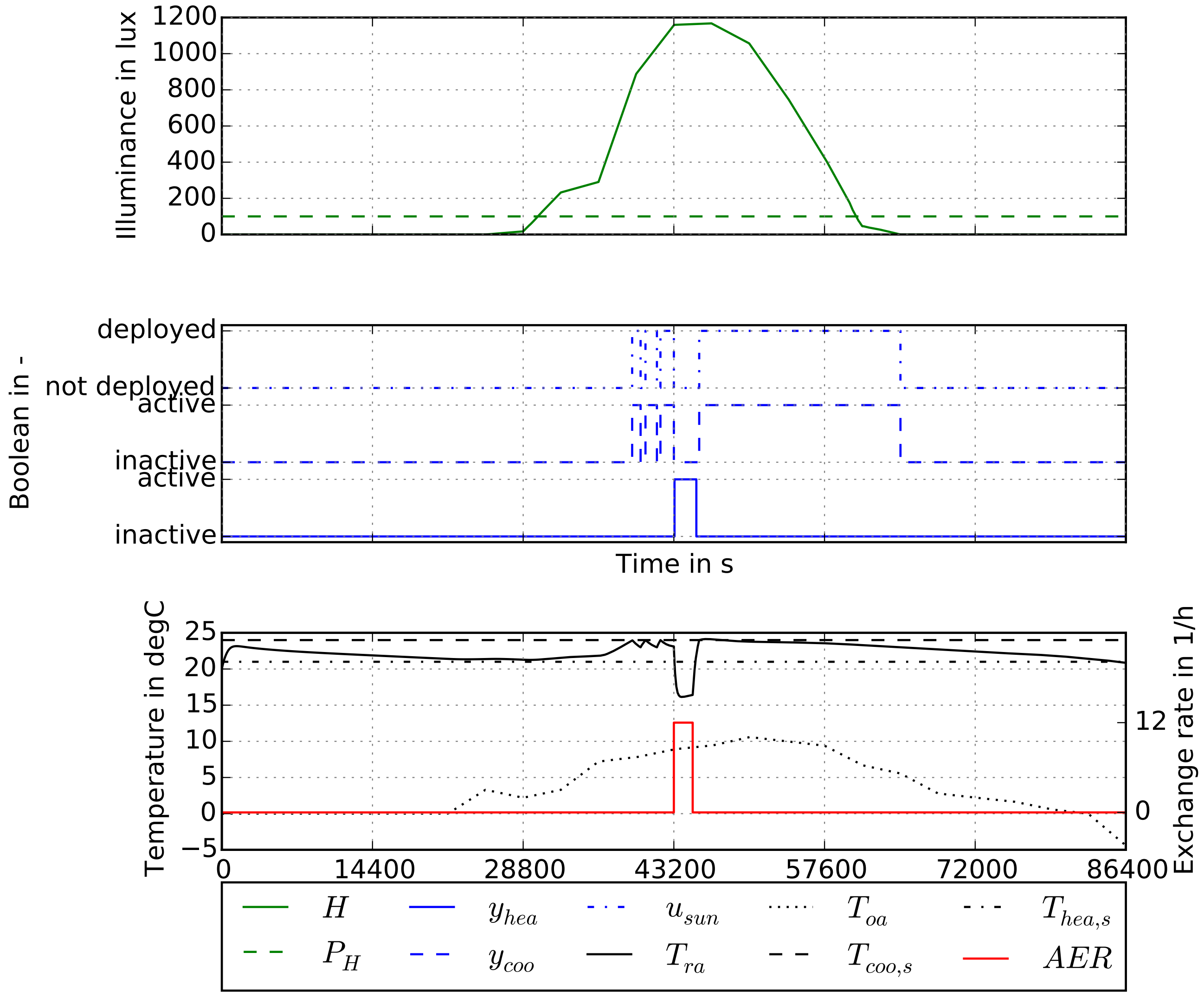

Figure 6. Simulation results of room automation example for one day. H - outdoor illuminance, P_H - threshold for illuminance, y_{hea} - heating mode, y_{coo} - cooling mode, u_{sun} - sunshade control signal, $T_{hea,s}$ - temperature set point heating, $T_{coo,s}$ - temperature setpoint cooling, T_{ra} - room air temperature, T_{oa} - outdoor air temperature, AER - air exchange rate.

and is prone for different interpretation. When implementing the models we use established documentation measures, namely UML class and activity diagrams to document our implementation. Having well-documented, commonly-agreed on simulation models of the control functions available as a reference implementation may help the wide-spread and further adoption of BAS in the buildings domain.

When modelling buildings, technical equipment and components and control logic of BAS the resulting hybrid system involves continuous and discrete event dynamics (Fritzson, 2014). In particular modelled discrete behaviour in control logic, e.g. a transition from one state to another if the condition $t_{Room} > 22°C$ is evaluated to true, triggers events involving state variables which need to be handled by the numerical solver. A large number of these events leads to a significant slow down of simulation speed when simulating the mentioned systems.

The Modelica modelling language provides built-in functionalities to efficiently handle events and should be applied when ever possible. Discrete behaviour with respect to time, e.g. sampling, can be efficiently handled using discrete variables or clocks introduced in Modelica 3.3 language specification (Otter et al., 2012).

The generation of state events can be prevented from using `noEvent(`*expression*`)` in case it is known that the respective expression is continuous and `smooth(`*p,expression*`)` if not known.

The use of clocked variables and expressions seems to be a promising path for efficient implementation of control behaviour in Modelica. In particular the ability to transfer clocked control systems from its Modelica implementation to clocked control hardware is a huge benefit. However, its effect on computational efficiency when simulating needs to be investigated as in the buildings domain distinguishing models with discrete and continuous dynamics is sometimes difficult; For example the discrete behaviour of a user opening a window when some temperature threshold is crossed may be modelled within the buildings model, thus discrete and continuous models are mixed.

5 Conclusion

In this work we present a modelling approach to model and simulate standardised control functions from building automation in Modelica. We exemplify this by modelling block like control functions from VDI 3813-2:2011 and state-centric control from VDI 3814-6:2009. In particular this includes models for sensor, actuator, operator- and display, application control functions and a template to model macro functions from VDI 3813-2:2011 and set of models to compose state graphs as specified in VDI 3814-6:2009 built on top of `StateGraph` package from Modelica Standard Library.

The usability of the models is demonstrated in two example applications linking a room automation solution to room models from `AixLib`-library (Constantin et al., 2014) and a state graph to control an air handling unit model from `Buildings`-library (Wetter et al., 2014).

The models presented, along with the models existing for building elements and equipment, allow to investigate the interaction and influences of an automation solution on the buildings behaviour in an integrated manner. Through the respective feedback from user models and also the interaction of user and automation solution as it is implemented in a real BAS is possible.

The total number of models and standards described included here is still limited. In future we plan to include more standardised, e.g. from ISO 16484:2011, and non-standardised control functions, e.g. for HVAC control.

We introduce the notion of a *semantic connector* which allows the library user to only connect control functions which are supposed to be connected, following the idea presented by Dibowski et al. (2010). The approach relies on a naming convention, despite the ability of consistency checking of Modelica modelling language for quantities and units. Future research may expand on this allowing to define variable semantics more freely as previously discussed and implemented by Leung et al. (2009).

In future we intend to investigate the potential benefits of using clocked variables in the definition of BAS control behaviour and standardised control functions for efficient simulation of the resulting hybrid systems.

Our intention is to stream line this effort with developments connected to the Annex60 effort and stipulate collaboration and reuse by open-sourcing the described models. For this purpose we intend to integrate the models within an open library. Through doing this we hope that this effort acts as a catalyst for implementing and providing control logic from BAS for building control in a comprehensive, well-documented and efficiently implemented way.

Acknowledgements

This research was performed as part of the Energie Campus Nürnberg and supported by funding through the 'Aufbruch Bayern (Bavaria on the move)' initiative of the state of Bavaria.

References

R. Baetens, R. De Coninck, J. Van Roy, B. Verbruggen, J. Driesen, L. Helsen, and D. Saelens. Assessing electrical bottlenecks at feeder level for residential net zero-energy buildings by integrated system simulation. *Applied Energy*, 96:74–83, 2012.

T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauß, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauß, D. Neumerkel, H. Olsson, and A. Viel. Functional mockup interface 2.0: The standard for tool independent exchange of simulation models. In *Proceedings of the International Modelica Conference*, pages 173–184, Munich, Germany, 2012.

M. Bonvini and A. Leva. A modelica library for industrial control systems. In *Proceedings of the International Modelica Conference*, pages 477–484, Munich, Germany, 2012.

E. Chrisofakis, A. Junghanns, C. Kehrer, and A. Rink. Simulation-based development of automotive control software with Modelica. In *Proceedings of the International Modelica Conference*, pages 1–7, Dresden, Germany, 2011.

A. Constantin, R. Streblow, and D. Müller. The Modelica *HouseModels* Library: Presentation and Evaluation of a Room Model with the ASHRAE Standard 140. In *Proceedings of the International Modelica Conference*, pages 293–299, Lund, Sweden, 2014.

H. Dibowski. *Semantischer Gerätebeschreibungsansatz für einen automatisierten Entwurf von Raumautomationssystemen*. PhD thesis, Department of Computer Science, TU Dresden, Dresden, Germany, 2013.

H. Dibowski, J. Ploennigs, and K. Kabitzsch. Automated Design of Building Automation Systems. *IEEE Transactions on Industrial Electronics*, 57(11):3606–3613, 2010.

Dymola, 2015. URL `http://www.3ds.com/products-services/catia/products/dymola`. Dassault Systemes AB, Lund, Sweden, [Accessed: 31-12-2016].

EN 15232:2013. Energy performance of buildings - Impact of Building Automation, Controls and Building Management, 2013.

P. Fritzson. *Principles of object-oriented modeling and simulation with Modelica 3.3: A cyber-physical approach*. John Wiley & Sons, 2014.

D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, 1987.

ISO 16484:2011. Building automation and control systems (BACS), 2011.

J. M.-K. Leung, T. Mandl, E. A. Lee, B. Osyk, C. Shelton, S. Tripakis, and B. Lickly. Scalable semantic annotation using lattice-based ontologies. In *Proceedings of MDELS*, pages 393–407, Denver, USA, 2009.

L. Liu. *Object-oriented Modeling and Efficient Simulation of C3-Systems*. PhD thesis, University of Saarland, Saarbrücken, Germany, 2013.

C. Nytsch-Geusen, J. Huber, M. Ljubijankic, and J. Rädler. Modelica BuildingSystems- eine Modellbibliothek zur Simulation komplexer energietechnischer Gebäudesysteme. *Bauphysik*, 35(1):21–29, 2013.

Object Management Group. OMG Unified Modeling Language, 2015.

M. Otter, B. Thiele, and H. Elmqvist. A Library for Synchronous Control Systems in Modelica. In *Proceedings of International Modelica Conference*, 2012.

G. F. Schneider, J. Oppermann, A. Constantin, R. Streblow, and D. Müller. Hardware-in-the-Loop-Simulation of a Building Energy and Control System to Investigate Circulating Pump Control Using Modelica. In *Proceedings of the International Modelica Conference*, pages 225–233, Versailles, France, 2015.

VDI 3813-2:2011. Building automation and control systems (BACS) Room control functions (RA functions), 2011.

VDI 3814-6:2009. Building automation and control systems (BACS) Graphical description of logic control tasks, 2009.

V. Venkatasubramanian, R. Rengaswamy, K. Yin, and S. N. Kavuri. A review of process fault detection and diagnosis: Part I: Quantitative model-based methods . *Computers & Chemical Engineering*, 27(3):293 – 311, 2003.

M. Wetter, W. Zuo, T. S. Nouidui, and X. Pang. Modelica Buildings Library. *Journal of Building Performance Simulation*, 7 (4):253–270, 2014.

M. Wetter, M. Fuchs, P. Grozman, L. Helsen, F. Jorissen, M. Lauster, D. Müller, C. Nytsch-Geusen, D. Picard, P. Sahlin, and M. Thorade. IEA EBC Annex 60 Modelica Library - An International Collaboration to Develop a Free Open-Source Model Library for Buildings and Community Energy Systems. In *BuiSim 2015*, Hyderabad, India, 2015.